INF574 project report

Marc Dufay, Lilian Bichot

December 2021

1 Introduction

1.1 Context

There exists many ways to create artistic imagery in 2D, but there is still a lack of effective ways to do it directly on 3D polygon meshes. So, in this project, based on article [2], we tackle an artistic issue : how to make meshes more cubic while preserving its details. Why cubic ? Because cubism is very present in art history. What do we mean by "preserve the details" ? We do not want to do on 3D meshes the equivalent of the pixelation of a 2D image, but instead we want to "cubify" only the core of the shape and preserve its superficial details. A quality challenge is to define a similarity metric aligned with human perception. An user interface challenge is to have nearly interactive performance on highly detailed meshes.

1.2 State of the art

1.2.1 Discriminative algorithms

In order to discern or transfer styles, some studies propose for instance to compare projected feature curves or sub-components of the shape. But we want to generate 3D stylized shapes directly, so we prefer generative algorithms.

1.2.2 Generative algorithms

There already exist many generative algorithms. One could think about voxelization, but it fails to preserve the details, and to correctly represent the global cube-shaping of the mesh. Huang uses a polycube method, but fails to preserve the superficial details. Liu uses image quantization, but fails to represent the global cube-shaping of the mesh.

1.3 Originality of our approach

Our algorithm requires no remeshing (except if we speed up the algorithm by collapsing edges at the beginning and adding faces at the end), but only the moving of the vertices. Thanks to well-known ARAP energy [3], our approach preserves the details of the original 3D mesh. Also, it generalizes well to polyhedral stylization (instead of simple cubic stylization).

2 Solving the problem

2.1 Mathematical description

2.1.1 Mathematical context

In order to cubify the mesh, we want to minimize a two-term energy. The first term is the well-known ARAP energy, and minimizing it tends to preserve the superficial details of the mesh. The second one represents the cubification of the mesh. Since we use l^1-norm, R_i tends to make $R_i.\hat{n}_i$ align with one of the x,y,z-axis.

Analytically, by choosing $\tilde{V}, \{R_i\}$, we want to minimize :

$$\sum_{i \in V} \sum_{j \in N(i)} \frac{w_{ij}}{2}.||R_i.d_{ij} - \tilde{d}_{ij}||_F^2 + \lambda.a_i.||R_i.\hat{n}_i||_1 \tag{1}$$

with V the original mesh, $\tilde{V}$ the cubified mesh, $N(i)$ the neighbors of vertex i, every R_i a rotation matrix, w_{ij} the cotangent weight and a_i the barycentric area, d_{ij} the edge vector between vertices i and j in the original mesh, $\tilde{d}_{ij}$ the same edge in the cubified mesh, and $\hat{n}_i$ the normal vector.

2.1.2 Algorithm

In order to minimize the energy, we apply a local step on each vertex until it converges, and then a global step on the mesh.

The equation of the local step is :

$$R_i^\star = argmin_{R_i \in SO(3)} \frac{1}{2}.||R_i.D_i - \tilde{D}_i||_{W_i}^2 + \lambda.a_i.||R_i.\hat{n}_i||_1 \tag{2}$$

with D_i and $\tilde{D}_i$ the matrices of the neighbors of vertex i in the original mesh and in the cubified mesh, and W_i the diagonal matrix of cotangent weights.

We can rewrite it as a equation system solvable by Alternating Direction Method of Multipliers [1], and then decompose it into an orthogonal Procrustes and a lasso problem (which we solve with a shrinkage step).

Let be, with $[z = R_i.\hat{n}_i]$ the constraint of the ADMM :

$$M_i = \begin{bmatrix} D_i & \hat{n}_i \end{bmatrix} . \begin{bmatrix} W_i & \\ & \rho^k \end{bmatrix} . \begin{bmatrix} \tilde{D}_i^T \\ (z^k - u^k)^T \end{bmatrix} \tag{3}$$

Then, by doing the singular value decomposition $[M_i = U_i.\Sigma_i.V_i^T]$, the updating of R_i^k is $R_i^{k+1} = V_i.U_i^T$, possibly changing the sign in order to keep $detR_i^{k+1} > 0$.

On the other hand, in order to solve the lasso problem, we update z^k to :

$$z^{k+1} = S_{\lambda.a_i/\rho^k}(R_i^{k+1}.\hat{n}_i + u^k) \tag{4}$$

with for each x, κ, j :

$$S_\kappa(x)_j = (1 - \frac{\kappa}{|x_j|})_+.x_j \tag{5}$$

Then, we possibly update the penalty ρ by multiplying it by a factor τ^{incr} or dividing it by a factor τ^{decr} depending on the predominance of the primal residual $||z^{k+1} - R_i^{k+1}.\hat{n}_i||$ or the dual residual $\rho.||z^{k+1} - z^k||$ among the other residual (by a μ ratio).

We run local steps until the primal residual **and** the dual residual are respectively smaller than $\epsilon^{primal} = \sqrt{3}.\epsilon^{abs} + \epsilon^{rel}.max\{||R_i^{k+1}.\hat{n}_i||, ||z^{k+1}||\}$ and $\epsilon^{dual} = \sqrt{3}.\epsilon^{abs} + \epsilon^{rel}.\rho.||u^{k+1}||$, with fixed ϵ^{abs} and ϵ^{rel}.

After having made converge every local step on every vertex, we run a global step, which consists in updating $\tilde{V}$ by solving $L.p = b$ with L the Laplace-Beltrami operator and b a vector which depends on the rotations. Because L is a sparse negative semi-define matrix, this can be done in a fast way using a Cholesky decomposition on sparse matrix.

In order to compute b, we initialize it to $0_{n,3}$ and we browse all the vertices i and all the neighbors j or the vertex i and then add to the i-th row of b :

$$-\frac{w_{i,j}}{2}.(R_i + R_j).d_{ij} \tag{6}$$

2.2 Architecture of the implementation

The main function creates an object from the class $CubicMinimizer$, and calls the function *single_step*. This builder initializes all the variables we need, and computes the normals, the neighbors, the distances to them thanks to the auxiliary function *compute_neighbour_dist*, the cotangent weights and the barycenter areas.

The function *single_step* calls the function *local_step* for each vertex of the mesh, and then calls the function *global_step*.

For each vertex of the mesh, the function *local_step*, during a maximum of -let's tell- 100 iterations successively (until the norm of primal and dual residuals are small enough) :

- Updates the optimal rotation matrices by solving the orthogonal Procrustes problem. In order not to recompute the same thing at each step, we decompose the matrix M_i between a fixed term and a variable term.
- Updates the rotated normal vectors by solving the lasso problem.
- Then updates the penalty and the scaled dual variable, especially by possibly increasing or decreasing the penalty depending on if the primal residual or the dual residual is greatly larger than the other residual, as seen before.

2.3 Efficiency

Based on the paper research [1], we set the value ϵ^{abs} to 10^{-5}, ϵ^{rel} to 10^{-3}, μ to 10, and τ^{incr} and τ^{decr} to 2. If λ is huge, in order to prevent the algorithm from reaching a local minima, we should increase the initial value of ϵ^{abs}.
A way to make the local step faster is to initialize u and ρ to the ones we got at the end of the previous iteration, and z to $R_i \cdot n_i$. This way, we start with values close to the optimum so we reduce the number of iterations needed.

3 Non-uniform lambdas

The paper suggests ways to alter the effects of the algorithm. One of them is by using different lambda values for different vertices. They do so in a few ways, like defining regions with different lambda values or looking at the normals.

But these methods do not allow the user to cubify only one part of the mesh with a smooth transition, and the controls to cubify one part are not intuitive enough. So we implemented a control which allows a user to specify for some vertices the lambda value they should have. Then the lambda values for all over vertices are defined looking at the fixed lambda values weighed by a function of the euclidean distance. The problem is that if we want to cubify the trunk of the rabbit but not its head, we would specify a small lambda for the forehead and a big lambda for the back, but since the extremity of one ear is closer from the back than from the forehead, it would be highly cubified, what we do not want. So we switched to a geodesic distance instead of a euclidean distance, and the result is much better.

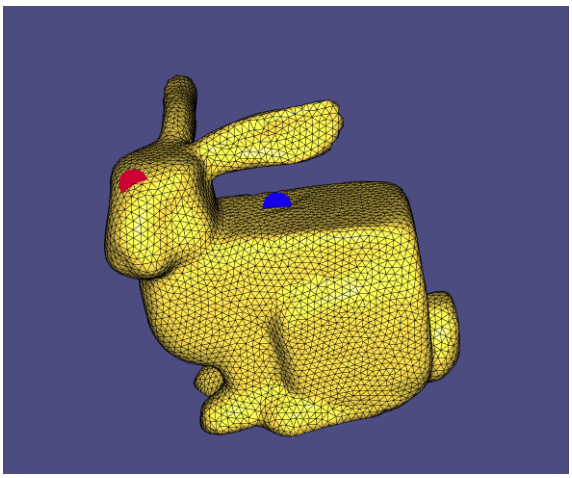


Figure 1: With euclidean distance

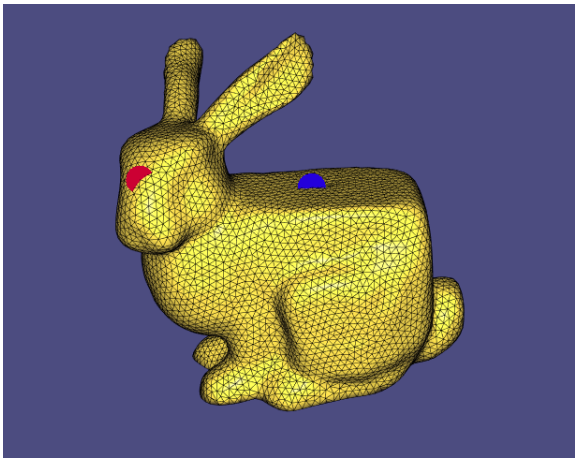
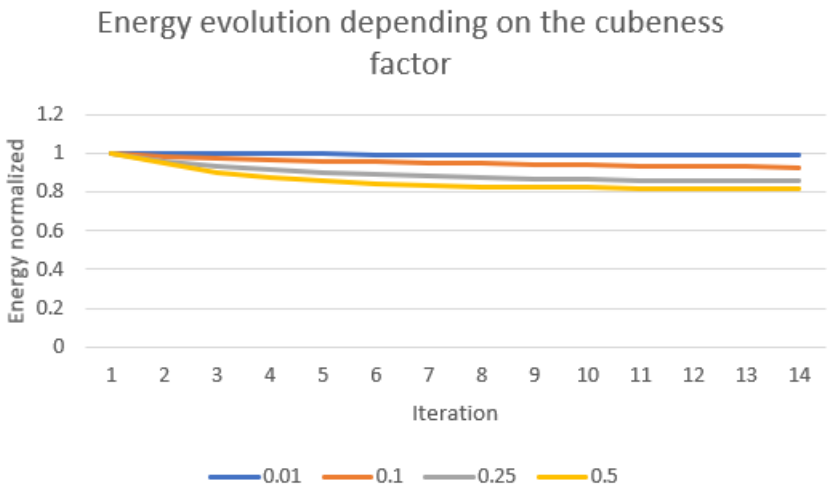


Figure 2: With geodesic distance

4 Polyhedral generalization

Our algorithm theoretically generalizes well to polyhedral style, by adding a matrix B in the energy expression. We can then solve it with a quadratic programming solver. We have tried to implement it following the pieces of advice from the main paper, but unfortunately it did not work. However, we put this implementation in our code, so you can have a look at it nevertheless.

5 Results



Our implementation works well. We see that the bigger the lambda, the faster the decreasing of the energy.

6 Extensions

- If we want a vertex i not to move during the cubification, we just need to set $\tilde{V}_i$ to a value. And if we want it -for instance- to stay on the yz-plane, we just need to set $(\tilde{V}_i)_x$ to 0.
- On the reasonable meshes we used, our algorithm was fast enough even in a laptop, so we did not felt the necessity to speed up the running. But if we wanted to, we could use Manson & Schaefer hierarchical approach : pre-process the mesh by edge-collapsing, then apply our algorithm, and then add back vertices in order to complexify back the cubified mesh.

References

[1] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. *Foundations and Trends in Machine Learning*, 3:1–122, January 2011.

[2] Hsueh-Ti Derek Liu and Alec Jacobson. Cubic Stylization. *ACM Transactions on Graphics*, 38(6):1–10, November 2019. arXiv: 1910.02926.

[3] Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In *Proceedings of the fifth Eurographics symposium on Geometry processing*, SGP '07, pages 109–116, Goslar, DEU, July 2007. Eurographics Association.